ID2208 Programming Web Services

Web Services Composition

Mihhail Matskin:
http://people.kth.se/~misha/ID2208/

Spring 2016

Content

Composition

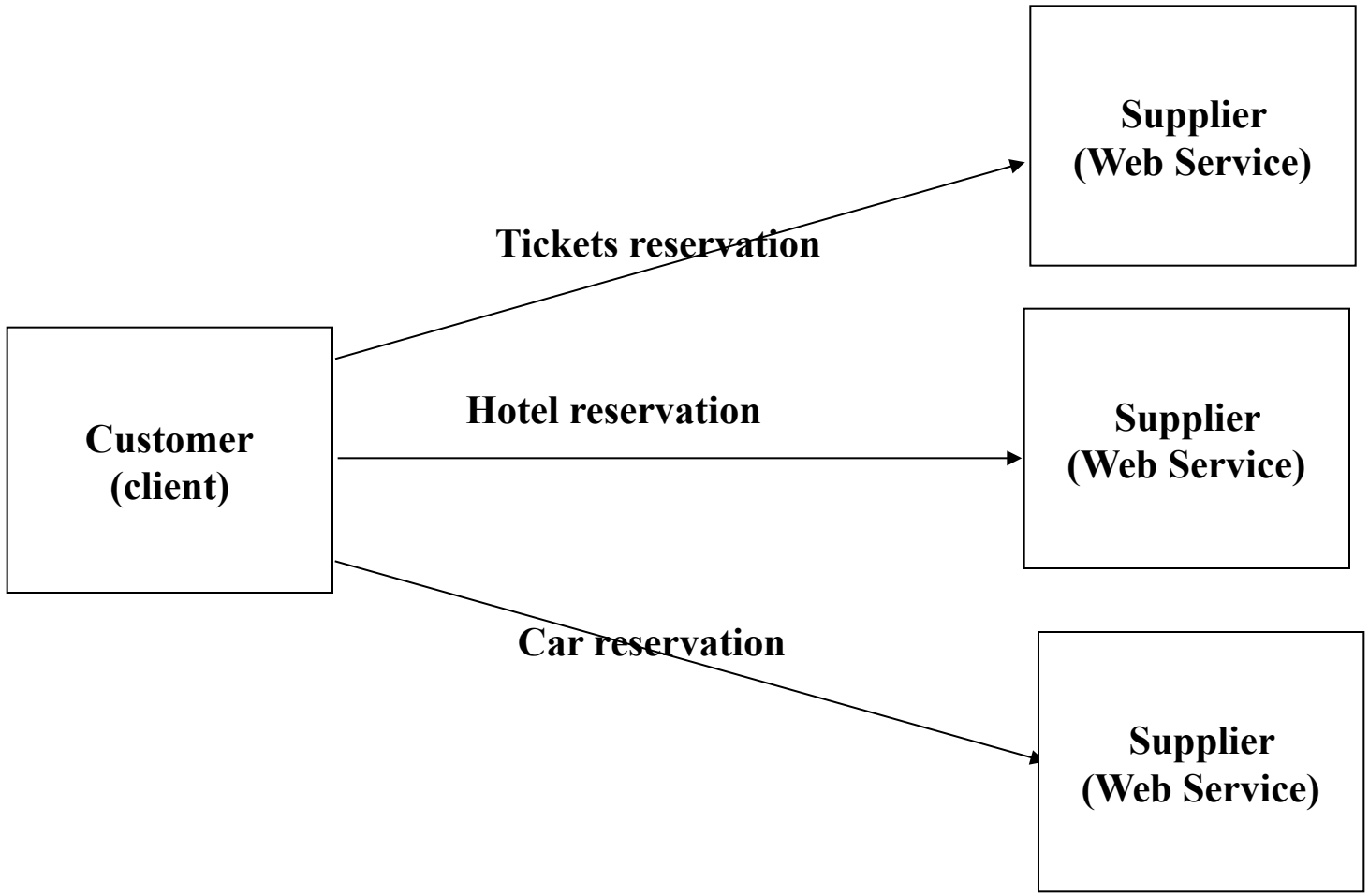
- The need for composition
- Composition vs. Coordination
- Service composition models
- WS-BPEL - Business Process Execution Language for Web Services

This lecture reference

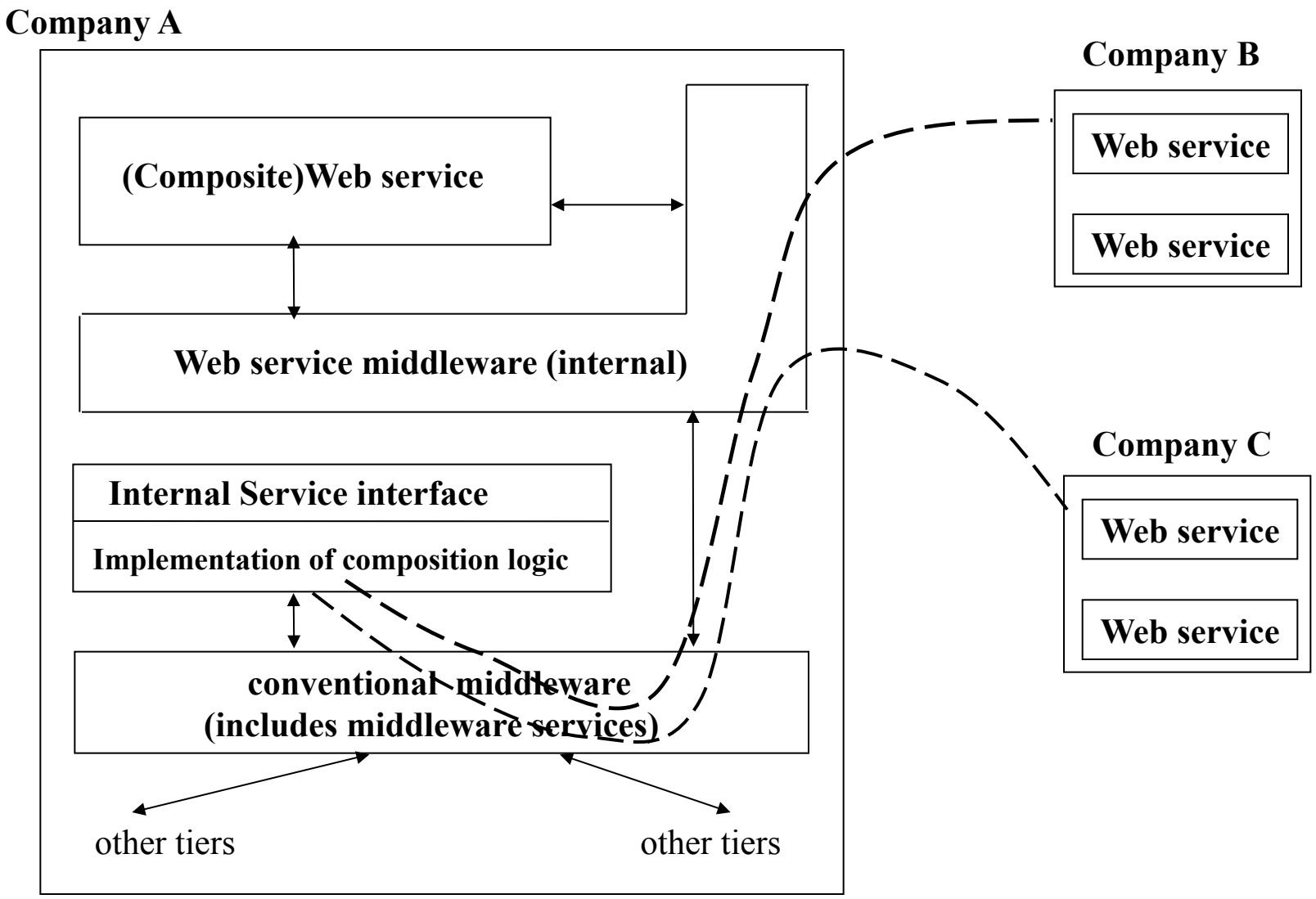
Text-book **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition**

Chapter 12

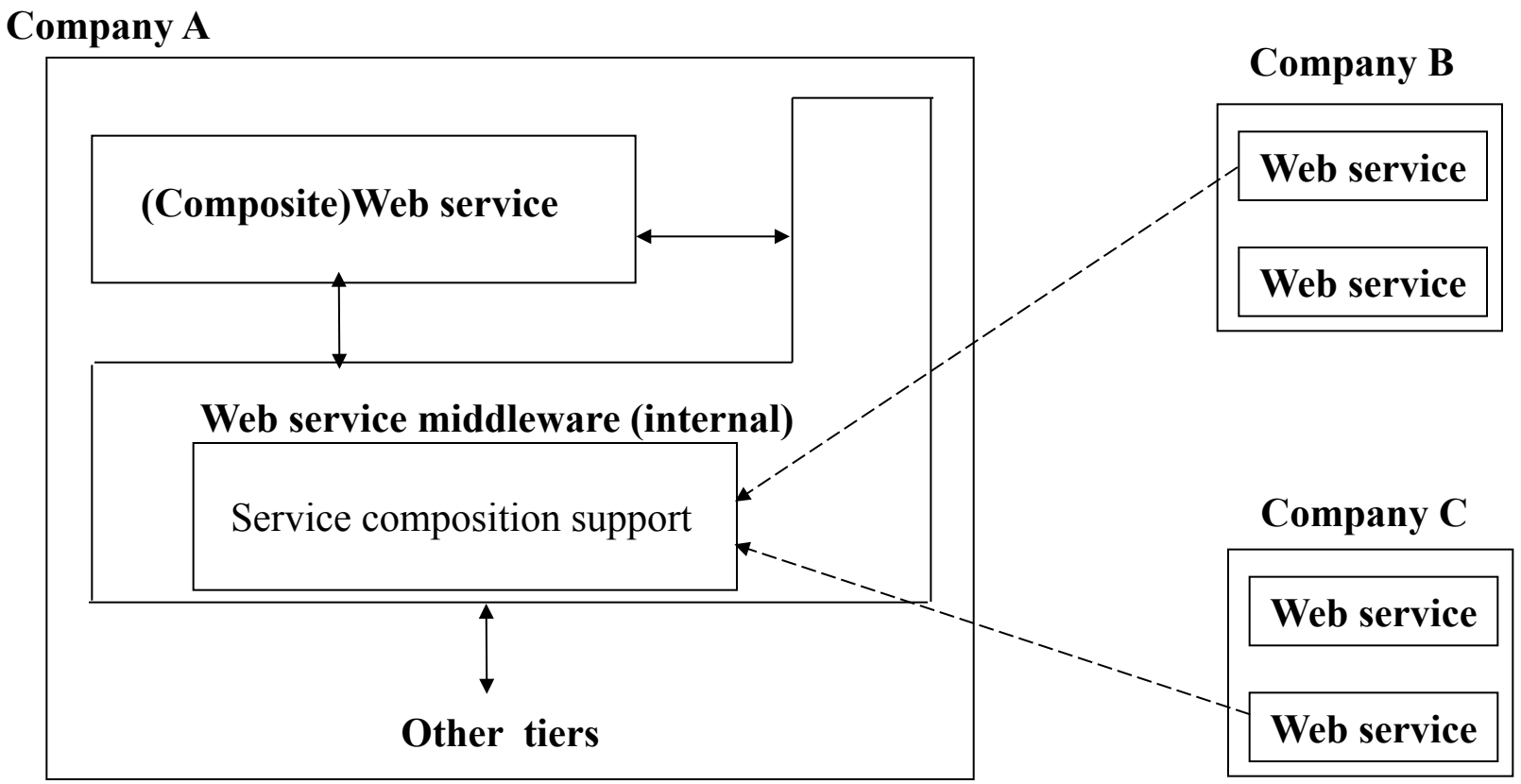
The Need for Composition



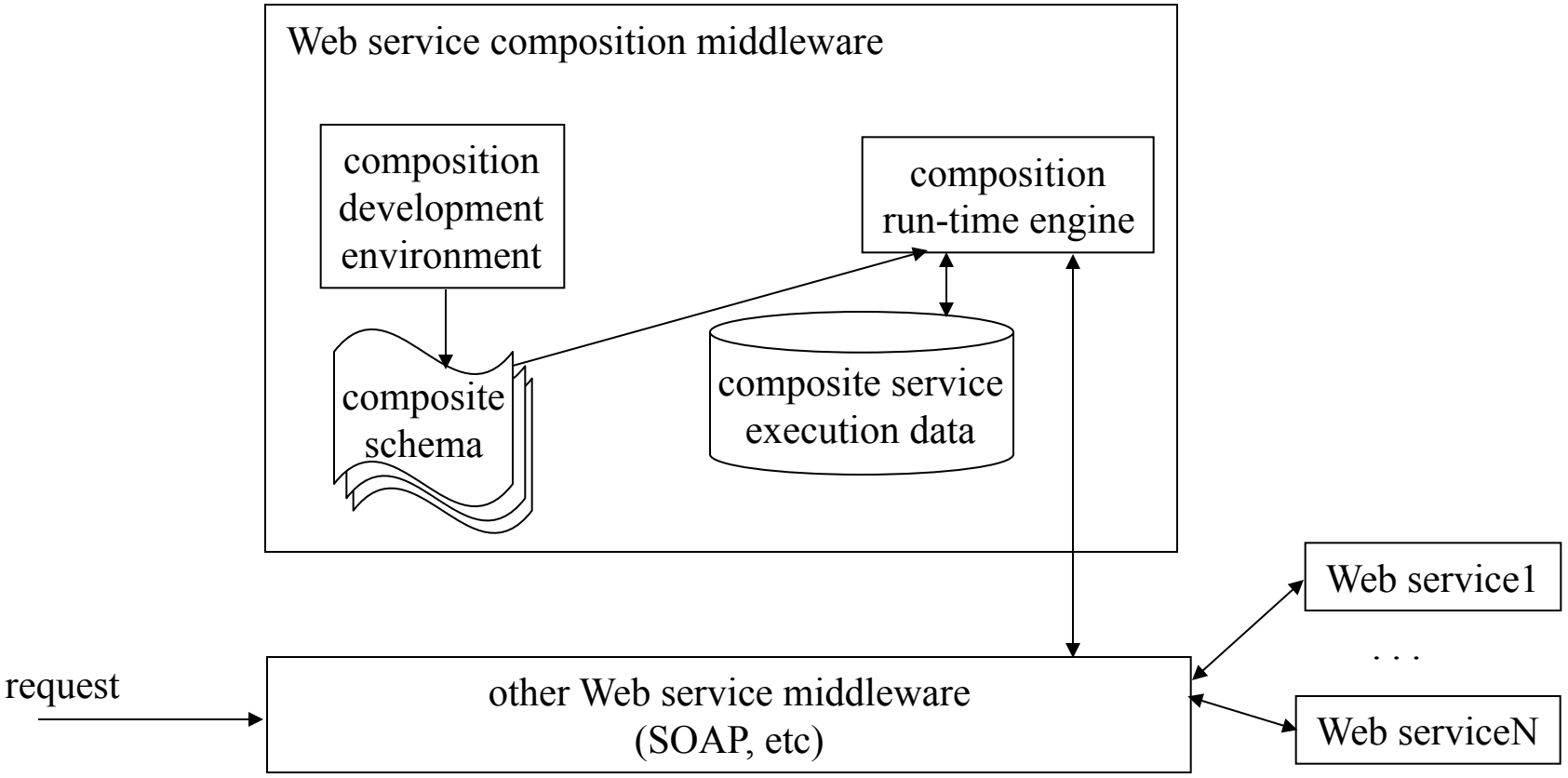
Conventional Middleware for Web Service Composition



Web Services Middleware for Web Service Composition



Elements of Composite middleware



Based on Alonso et al. "Web services. Concepts, Architectures and Applications"

Composition vs. Coordination

- Composition is about the internal implementation of services
- Composition specification is for consumption of Web service middleware which automates execution of services
- Whether a service is basic or composite is irrelevant from the client's perspective
- Coordination concerns with external/conversational aspects of Web services, public conversation protocols
- Conversations compliant with a coordination protocol are supported by conversation controllers which purpose is not to execute any business logic but to dispatch messages to internal objects and to verify protocol compliance
- The conversation controller could be unaware of whether it is dispatching messages to a basic or a composite service

Composition and Coordination

- The coordination protocol imposes requirements on how the composition is to take place
- The composition logic determines the conversations that a composite service is able to execute
- There is a correlation between a Web service's internal composition and external coordination

Programming model for composition

- Two-levels programming (Software Engineering)
  - Programming in the large
  - Programming in the small
- Two-levels programming (Web services)
  - Process model (process definition)
  - Service implementation (process instance)

Service Composition Models (dimensions)

- Component model
- Orchestration model
- Composite schema creation mechanism
- Data and data access model
- Service selection model
- Transactions
- Exception handling

BPEL: Business Process Execution Language for Web Services

- BPEL is a language that can support the specifications of both composition schemas and coordination protocols
- BPEL specifications are XML documents that define different aspects of a process
- BPEL composition schemas are executable process specifications that define implementation logic of a composite process

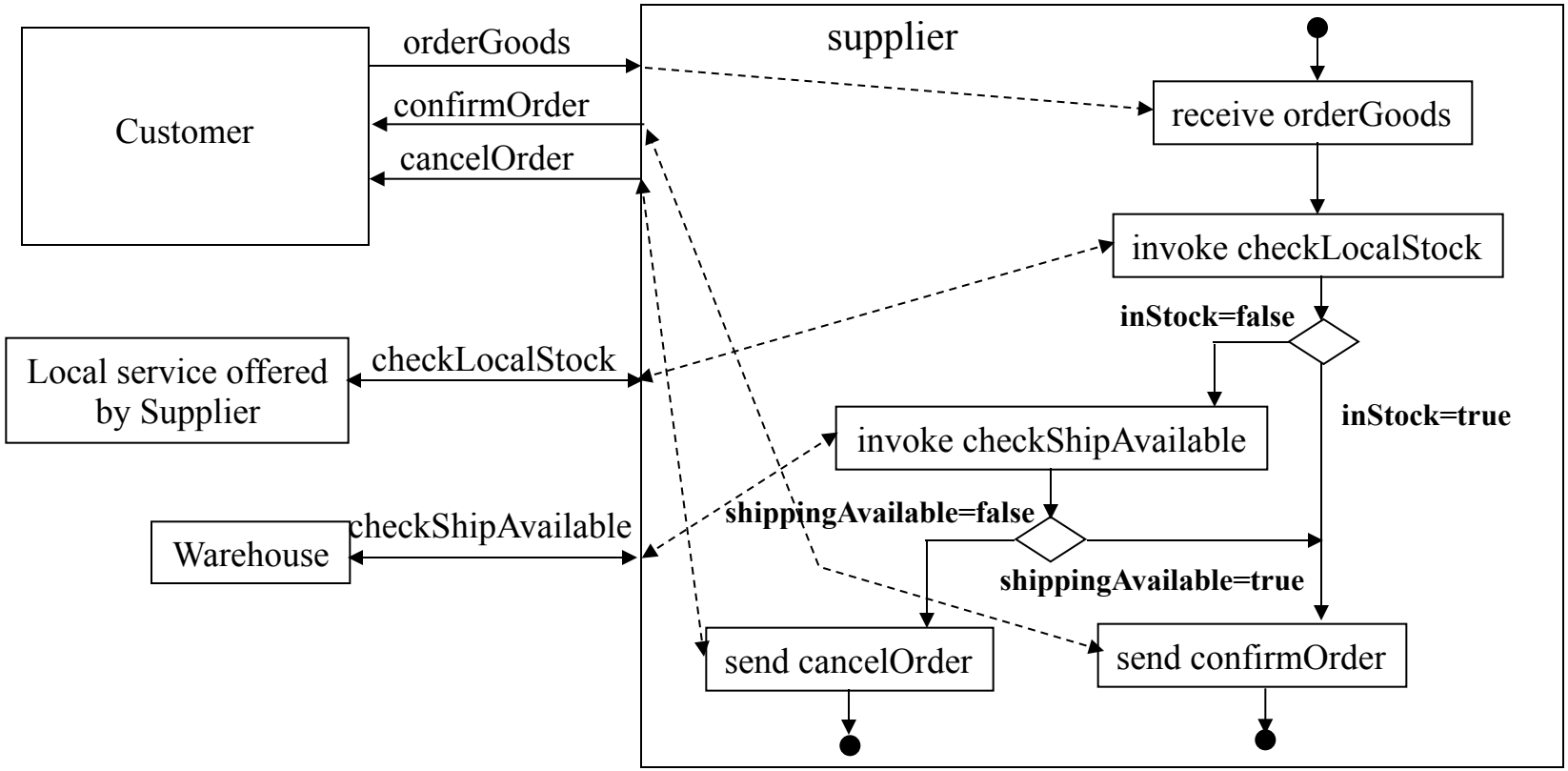
BPEL: Component Model

- BPEL's component model consists of activities which can be basic or structured
- Basic activities represent the actual components and correspond to the invocation of WSDL operation
- BPEL assumes that the interfaces of the interacting Web services are defined in terms of WSDL port types and interact by exchanging messages

Orchestration model

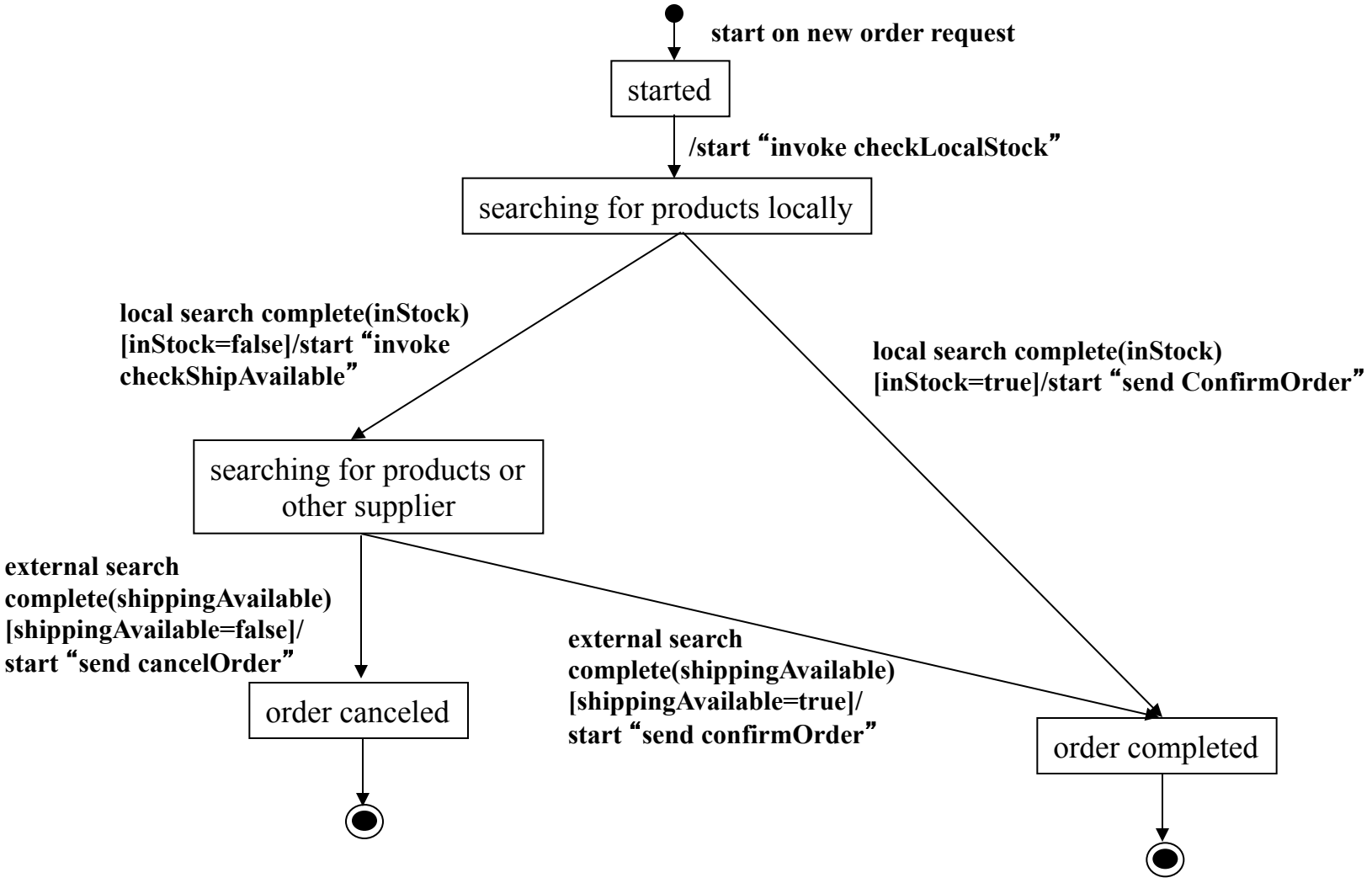
- Orchestration deals with how different services are composed into a coherent whole
- It specifies the order to invoke services
- Alternative models
  - Activity diagram
  - Statecharts
  - Petri nets
  - π - calculus
  - Activity hierarchies
  - Rule-based Orchestration

Orchestration model (Activity diagrams)



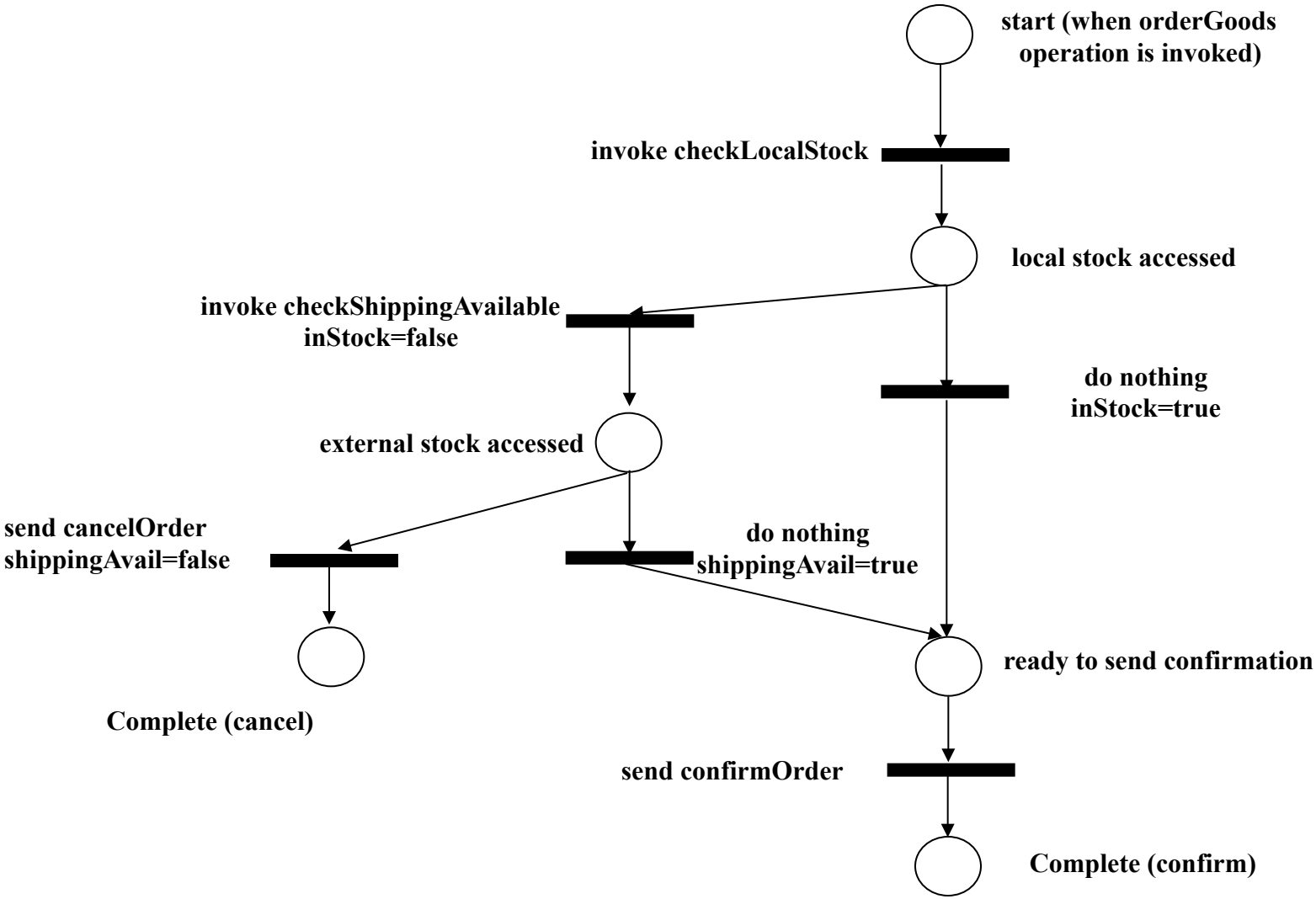
Based on Alonso et al. "Web services. Concepts, Architectures and Applications"

Orchestration model (Statecharts)



Based on Alonso et al. "Web services. Concepts, Architectures and Applications"

Orchestration model (Petri Nets)



Orchestration model (π - calculus)

- *[var=value]P* - *P* is executed iff *var=value*
- *P.P* - two processes execute in sequence.
- *P|P* - two processes execute in parallel.
- *P + P* - represents a non-deterministic choice which either the first process or the second process will execute.
- *0* - an inactive process and it does not perform any action
- *a(x).P* - An input prefixed process *a(x).P* receives a variable or message *x* through port *a* then executes process *P*
- $\bar{a}\langle x_i \rangle.P$ - emits message *x* at port *a* then executes process *P*.
- *(νa)P* - defines a name *a* local to *P*. Unlike the global name, the name *a* is private and its scope is limited to *P*.
- *!a(x).P* - stands for a countably infinite number of copies of channel a in parallel.

Orchestration model (π-calculus)

- Commutative law:

  $P|Q \equiv Q|P$ $\qquad$ $P + Q \equiv Q + P$
- Associative law:

  $(P|Q)|R \equiv P|(Q|R)$ $\qquad$ $(P + Q) + R \equiv P + (Q + R)$
- Inactive law:

  $P \equiv P|0 \equiv P + 0 \equiv P.0 \equiv 0.P$
- Replication law:

  $!a(x).P \equiv a(x).P|!a(x).P$

Orchestration model (π-calculus)

P1= receiveOrderGoods.invokeCheckLocalStock

P2=[shippingArrival=false]sendCancelOrder+
[shippingArrival=true]sendConfirmOrder

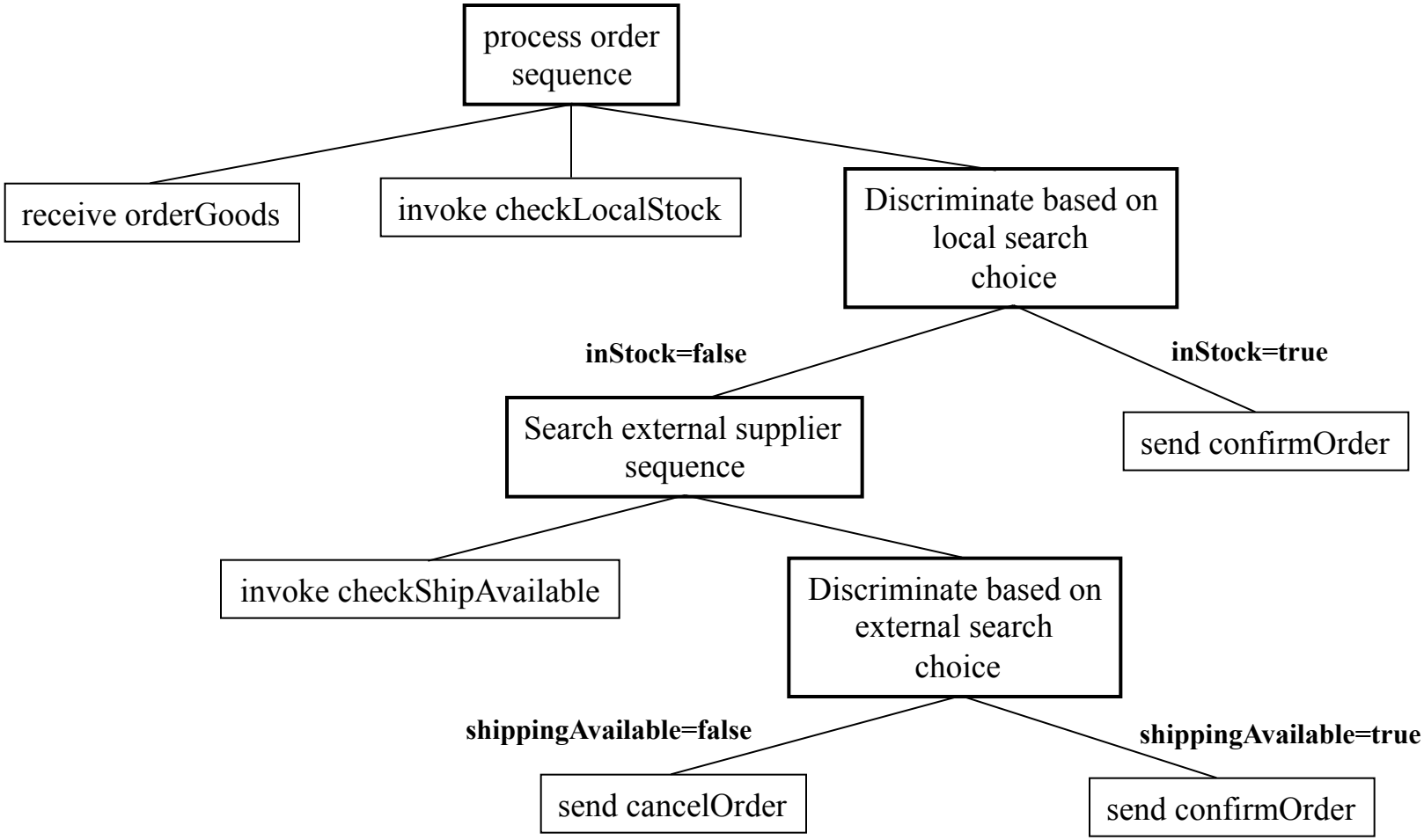
P3=invokeCheckShipAvailable.P2

Procurement=P1.([inStock=false]P3+
[inStock=true]sendConfirmOrder)

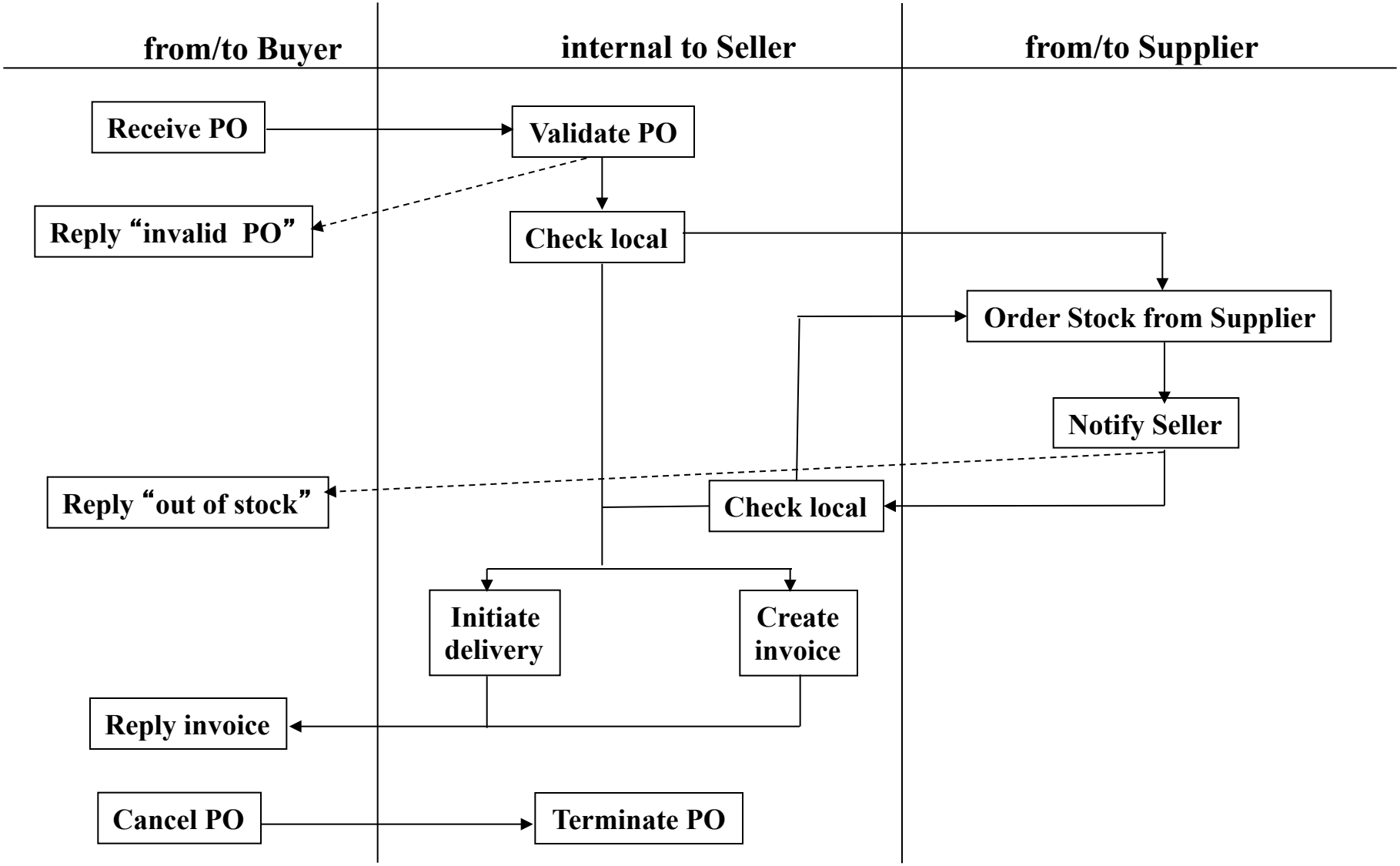
Rule-based Orchestration

```
ON receive orderGoods
IF true
THEN invoke checkLocalStock;

ON complete(checkLocalStock)
IF (inStock==true)
THEN send confirmOrder;

ON complete(checkLocalStock)
IF (inStock==false)
THEN invoke checkShipAvailable;

ON complete(checkShipAvailable)
IF (shipAvail==false)
THEN send confirmOrder;

ON complete(checkShipAvailable)
IF (shipAvail==false)
THEN send cancelOrder;
```

Activity hierarchies



Business Process: SkatesTown Example



PortTypes and Messages

```
<!-- Message definitions -->
    <message name="poSubmissionRequest">
        <part name="purchaseOrder" element="po:po"/>
    </message>
    <message name="poSubmissionResponse">
        <part name="invoice" element="inv:invoice"/>
    </message>
    <message name="poSubmissionFaultInvalidPO">
        <part name="customerID" element="xsd:ID"/>
        <part name="orderNumber" element="xsd:positiveInteger"/>
    </message>
    <message name="poSubmissionFaultOutOfStock">
        <part name="customerID" element="xsd:ID"/>
        <part name="orderNumber" element="xsd:positiveInteger"/>
    </message>
    <message name="cancelPurchaseOrderRequest">
        <part name="purchaseOrder" element="po:po"/>
    </message>
```

PortTypes and Messages

```
<!-- Port type definitions -->
 <portType name="poSubmissionPortType">
   <operation name="doSubmission">
     <input message="pos:poSubmissionRequest"/>
     <output message="pos:poSubmissionResponse"/>
     <fault name="invalidPO"
            message="pos:poSubmissionFaultInvalidPO"/>
     <fault name="outOfStock“
            message="pos:poSubmissionFaultOutOfStock"/>
   </operation>


   <operation name="cancelPurchaseOrder">
     <input message="pos:cancelPurchaseOrderRequest"/>
   </operation>
 </portType>
```

PortTypes and Messages

```
<portType name="orderSuppliesPortType">
   <operation name="orderSupplies">
     <input message="asp:orderSuppliesRequest"/>
   </operation>
</portType>
```

```
<portType name="orderSuppliesCallbackPortType">
   <operation name="orderSuppliesOK">
     <input message="asp:orderSuppliesResponse"/>
   </operation>
   <operation name="orderSuppliesFailed">
     <input message="asp:orderSuppliesFault"/>
   </operation>
 </portType>
```

WSDL extensions (partner link type)

```
<plnk:partnerLinkType
      name="purchaseOrderPartnerLinkType">
  <plnk:role name="seller">
      <plnk:portType name="pos:poSubmissionPortType"/>
  </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType
      name="purchaseOrderSuppliesPartnerLinkType">
  <plnk:role name="seller">
      <plnk:portType name="sup:orderSuppliesCallbackPortType"/>
  </plnk:role>
  <plnk:role name="supplier">
      <plnk:portType name="sup:orderSuppliesPortType"/>
  </plnk:role>
</plnk:partnerLinkType>
```

Process Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="purchaseOrderProcess" . . .>
 <!-- Partner link definitions -->
  <partnerLinks>
   <partnerLink name="supplier" myRole="seller" partnerRole="supplier"
      partnerLinkType="plt:purchaseOrderSuppliesPartnerLinkType"/>
   <partnerLink name="buyer" myRole="seller"
      partnerLinkType="plt:purchaseOrderPartnerLinkType"/>
         . . .
  </partnerLinks>
           . . .

  <!-- Variable definitions -->
  <variables>
   <variable name="poSubmissionRequest"
        messageType="pos:poSubmissionRequest"/>
         . . .
  </variables>
  <!-- Correlation set definitions -->
  <correlationSets>
   <correlationSet name="orderCorrelationSet"
        properties="ppa:customerID ppa:orderNumber"/>
  </correlationSets>
  <!-- Fault handler definitions -->
  <faultHandlers>. . . </faultHandlers>
  <!-- Event handler definitions -->
  <eventHandlers>. . .</eventHandlers>
  <!-- Activity definitions -->
  <sequence>
   <!-- activity1 -->
         . . .
  </sequence>
</process>
```

BPEL: Process lifecycle

- Process lifecycle begins with creation of process instance
  - it happens implicitly on receipt of a message
- The lifecycle ends either upon successful completion or by termination
  - termination (by cancellation or by fault) is abnormal end

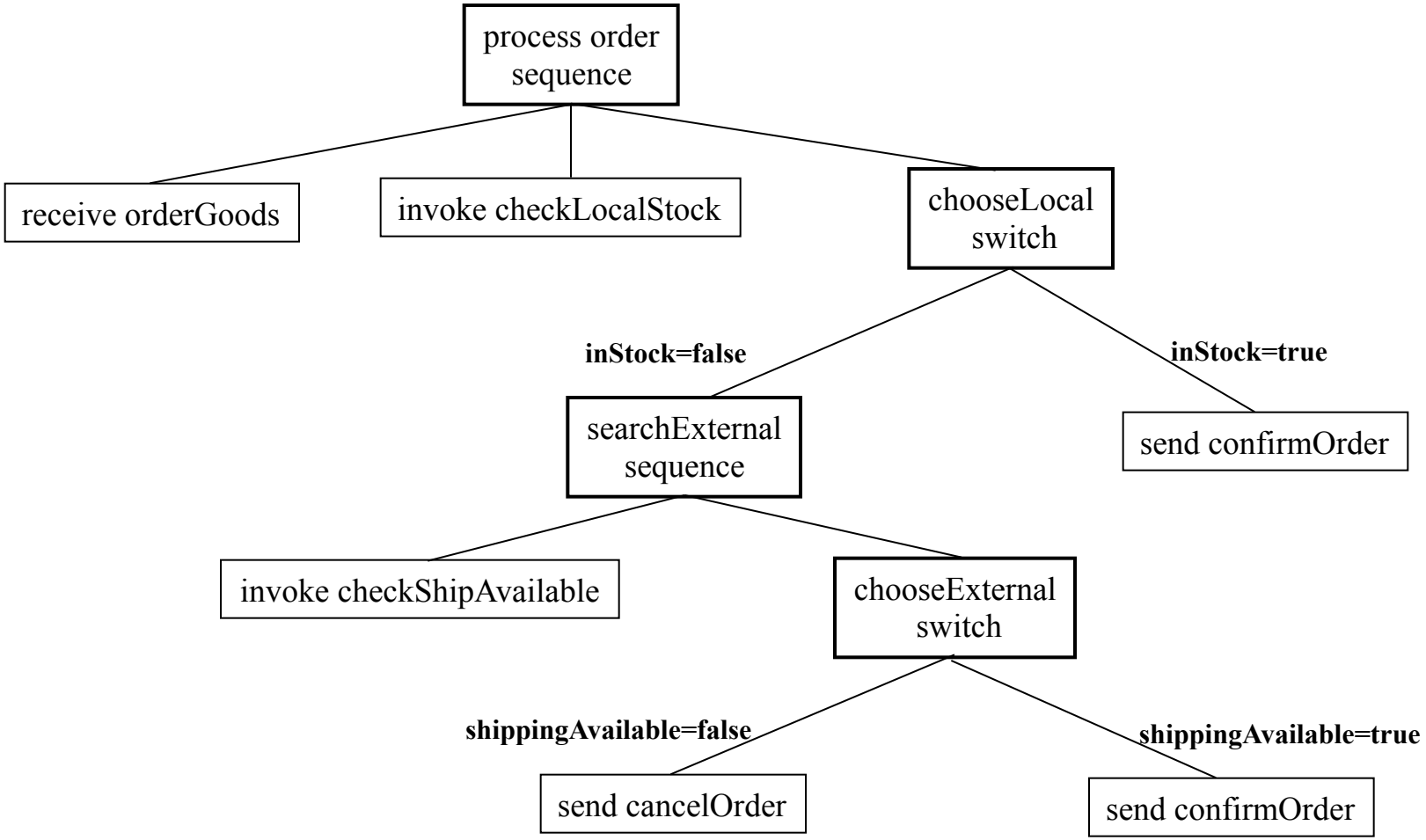
BPEL: Orchestration Model
Basic Activities

- **`receive`** – waits to receive a request message from partner
- **`reply`** – responses to **`receive`** message
- **`invoke`** - invokes an operation on a **`portType`** provided by another partner
- **`terminate`** – terminates the process
- **`throw`** - exceptional situation by throwing a fault
- **`assign`** – data manipulation with variables
- **`empty`** – no-op instruction
- **`compensate`** – triggers compensation for successfully completed group of activities

BPEL: Orchestration Model (structured activities)

- **`sequence`** – a set of activities to be executed sequentially
- **`switch`** – similar to C or Java
- **`pick`** – includes a set of events each associated with an activity
- **`while`** – includes exactly one activity, which is executed repeatedly while the condition is true
- **`flow`** – groups a set of activities to be started in parallel
- **`scope`** – lets you defines nested activities

BPEL: Orchestration Model (basic and structured activities)



Data Types and Data transfer

- BPEL maintains the state of the process and manipulates control data by means of variables
- Once defined, variables can be used as input or output parameters of operation invocations or referred to by conditions
- Initialization upon completion of activity (output parameters of receive or invoke activities) or explicitly by **`assign`** activity

BPEL: Data handling

```
<variables>
        <variable name="poSubmissionRequest"
                messageType="pos:poSubmissionRequest"/>
        <variable name="poSubmissionResponse"
                messageType="pos:poSubmissionResponse"/>
        <variable name="poSubmissionFaultInvalidPO"
                messageType="pos:poSubmissionFaultInvalidPO"/>
        <variable name="poSubmissionFaultOutOfStock"
                messageType="pos:poSubmissionFaultOutOfStock"/>
        <variable name="cancelPurchaseOrderRequest"
                messageType="pos:cancelPurchaseOrderRequest"/>
        <variable name="validatePurchaseOrderResponse"
                messageType="skt:validatePurchaseOrderResponse"/>
        <variable name="checkLocalStockResponse"
                messageType="skt:checkLocalStockResponse"/>
        <variable name="orderSuppliesRequest"
                messageType="sup:orderSuppliesRequest"/>
        <variable name="orderSuppliesResponse"
                messageType="sup:orderSuppliesResponse"/>
        <variable name="orderSuppliesFault"
                messageType="sup:orderSuppliesFault"/>
</variables>
```

BPEL: Data handling (**assign** activity)

```
<assign>
   <copy>
      <from variable="poSubmissionRequest"/>
       <to variable="trueCopyOfPoSubmissionRequest"/>
   </copy>
   <copy>
       <from variable="poSubmissionRequest"
             part="purchaseOrder"/>
       <to variable="orderSuppliesRequest"
             part="orderSupplies"/>
   </copy>
   <copy>
       <from partnerLink="supplier"
             endpointReference="myRole"/>
       <to variable="orderSuppliesRequest"
             part="endpointReferenceOfSeller"/>
   </copy>
</assign>
```

BPEL: Data handling (**assign** activity)

```
<assign>
   <copy>
        <from variable="poSubmissionRequest"
              part="purchaseOrder" query="/billTo/id"/>
        <to variable="poSubmissionFaultInvalidPO"
              part="customerID"/>
   </copy>
   <copy>
        <from variable="poSubmissionRequest"
              part="purchaseOrder" query="/id"/>
        <to variable="poSubmissionFaultInvalidPO"
              part="orderNumber"/>
   </copy>
</assign>
```

BPEL: Data handling (**assign** activity)

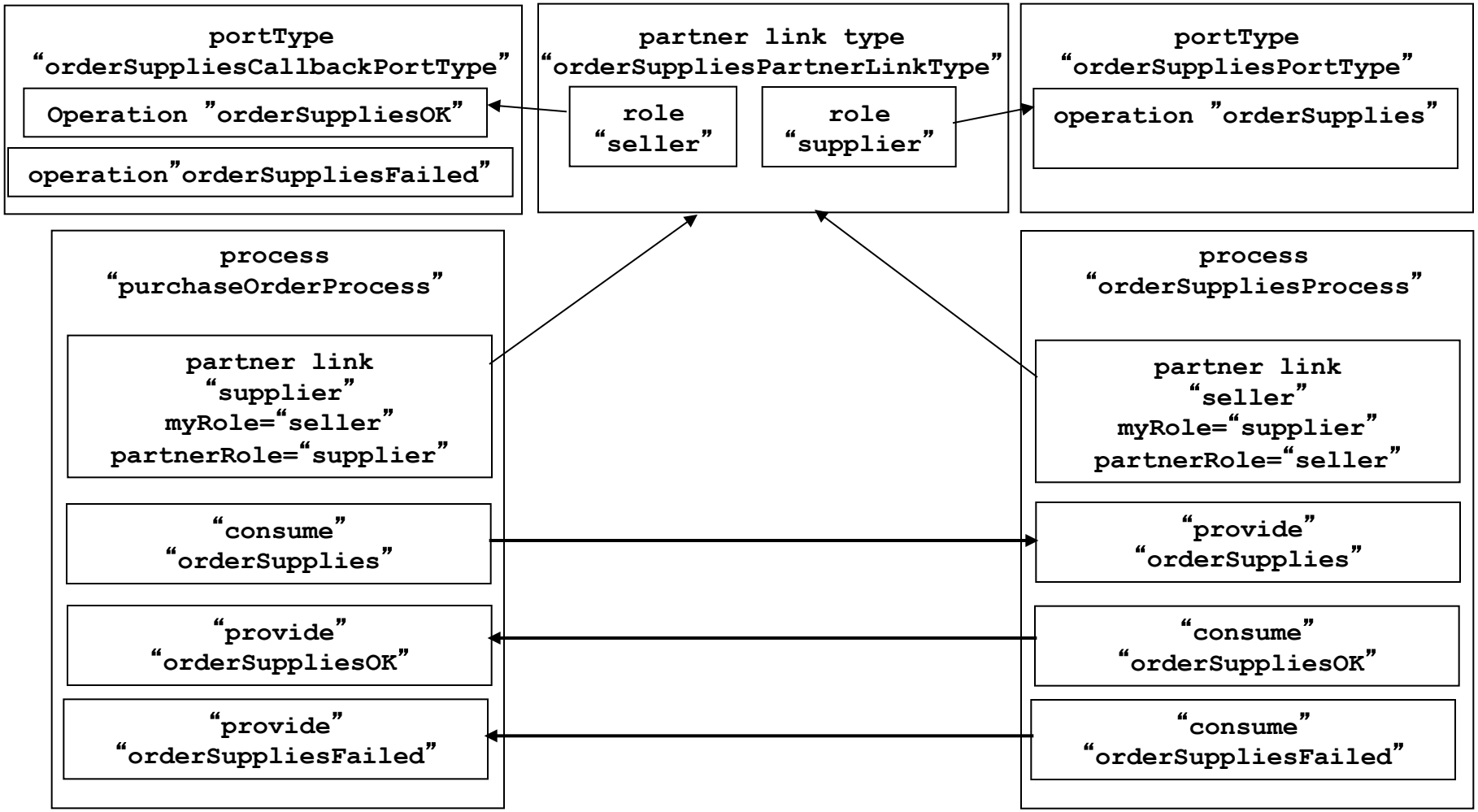
```
<assign>
   <copy>
        <from><xsd:integer>1</xsd:integer></from>
        <to variable="itemCount" />
   </copy>
</assign>
```

```
<assign>
   <copy>
        <from
           expression="bpws:getVariableData("itemCount") + 1" />
        <to variable="itemCount" />
   </copy>
</assign>
```

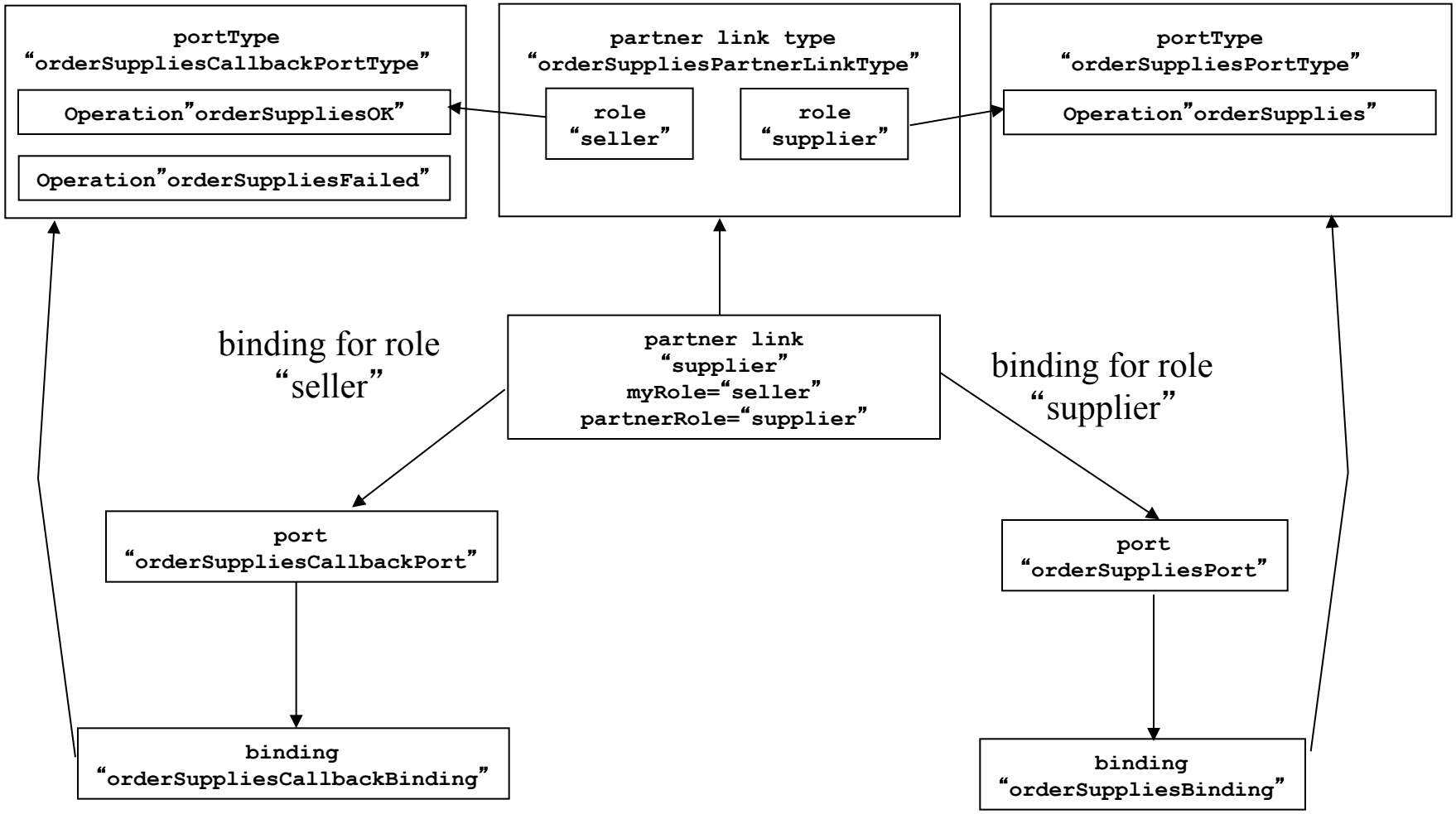
Service Selection (partner links)

```
<portType name="orderSuppliesPortType">
        <operation name="orderSupplies">
              <input message="asp:orderSuppliesRequest"/>
       </operation>
</portType>
<portType name="orderSuppliesCallbackPortType">
       <operation name="orderSuppliesOK">
              <input message="asp:orderSuppliesResponse"/>
       </operation>
        <operation name="orderSuppliesFailed">
               <input message="asp:orderSuppliesFault"/>
        </operation>
</portType>
<plnk:partnerLinkType  name="orderSuppliesPartnerLinkType">
       <plnk:role name="seller">
              <plnk:portType name="sup:orderSuppliesCallbackPortType"/>
       </plnk:role>
       <plnk:role name="supplier">
              <plnk:portType name="sup:orderSuppliesPortType"/>
       </plnk:role>
</plnk:partnerLinkType>
<partnerLink name="supplier"
       partnerLinkType name="plnt:orderSuppliesPartnerLinkType"
       MyRole="seller" partnerRole="supplier">
```

Service Selection (partner links)



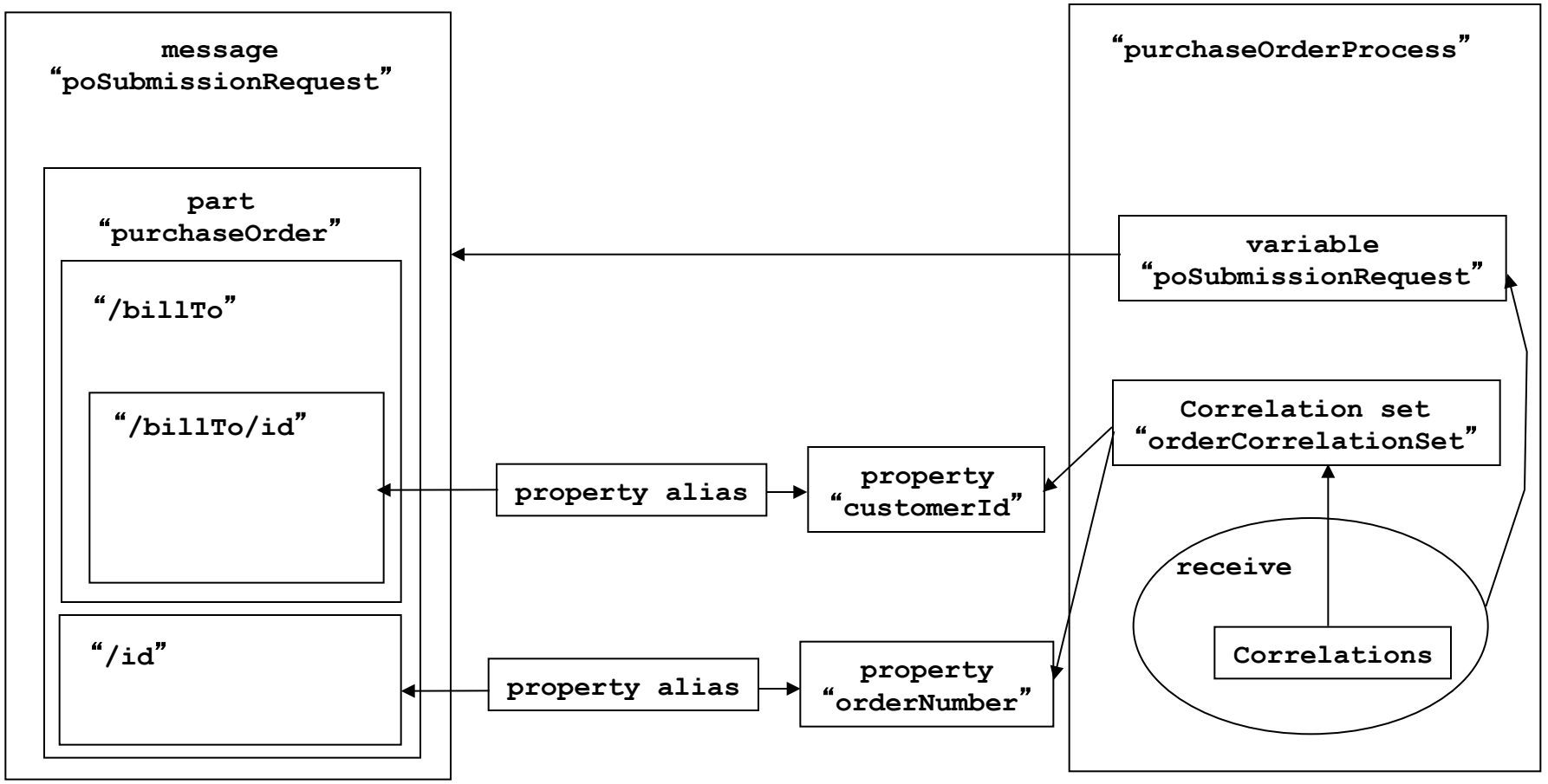
Service Selection (partner link binding)



Service Selection (partner link binding)

- Dynamic binding via endpoint references
- Endpoint references can be dynamically assigned to or from message parts

Service Selection (properties and correlation sets)



Service Selection (properties and correlation sets)

```
<!-- Property definitions -->
  <wsbp:property name="customerID" type="xsd:ID"/>
  <wsbp:property name="orderNumber" type="xsd:positiveInteger"/>
<!-- Property alias definitions -->
  <wsbp:propertyAlias propertyName="customerID" part="purchaseOrder"
       messageType="pos:poSubmissionRequest" query="/billTo/id"/>
  <wsbp:propertyAlias propertyName="orderNumber" part="purchaseOrder"
       messageType="pos:poSubmissionRequest" query="/id"/>
  <wsbp:propertyAlias propertyName="customerID" part="invoice"
       messageType="pos:poSubmissionResponse" query="/billTo/id"/>
  <wsbp:propertyAlias propertyName="orderNumber" part="invoice"
       messageType="pos:poSubmissionResponse" query="/id"/>
  <wsbp:propertyAlias propertyName="customerID"
      messageType="pos:poSubmissionFaultInvalidPO" part="customerID"/>
  <wsbp:propertyAlias propertyName="orderNumber"
     messageType="pos:poSubmissionFaultInvalidPO" part="orderNumber"/>
  <wsbp:propertyAlias propertyName="customerID"
       messageType="pos:poSubmissionFaultOutOfStock" part="customerID"/>
  <wsbp:propertyAlias propertyName="orderNumber"
    messageType="pos:poSubmissionFaultOutOfStock" part="orderNumber"/>
  <wsbp:propertyAlias propertyName="customerID" part="purchaseOrder"
     messageType="pos:cancelPurchaseOrderRequest" query="/billTo/id"/>
  <wsbp:propertyAlias propertyName="orderNumber" part="purchaseOrder"
       messageType="pos:cancelPurchaseOrderRequest" query="/id"/>
<!-- Correlation set definitions -->
  <correlationSets>
    <correlationSet name="orderCorrelationSet"
       properties="customerID orderNumber"/>
  </correlationSets>
```

Invoking Web Services Operations (**invoke** activity)

```
<invoke partnerLink="local"
          portType="skt:skatestownPortType"
        operation="checkLocalStock"
        inputVariable="poSubmissionRequest"
        outputVariable="checkLocalStockResponse">
    <correlations>
        <correlation set="orderCorrelationSet" initiate="no"
              pattern="out">
        <correlation set="anAdditionalSet" initiate="yes"
              pattern="out">
    </correlations>
</invoke>
```

Providing Web Services Operations (`receive/reply` activity)

```
<receive partnerLink="buyer"
        portType="pos:poSubmissionPortType“
        operation=“doSubmission"
        createInstance="yes"
        variable="poSubmissionRequest">
   <correlations>
      <correlation set="orderCorrelationSet" initiate="yes"/>
   </correlations>
</receive>


<reply partnerLink="buyer"
        portType="pos:poSubmissionPortType"
        operation=“doSubmission"
        variable="poSubmissionResponse">
   <correlations>
      <correlation set="orderCorrelationSet“ initiate="no"/>
   </correlations>
</reply>
```

Providing Web Services Operations (**pick** activity)

```
<pick>
   <onMessage partnerLink="supplier"
        portType="sup:orderSuppliesCallbackPortType"
        operation="orderSuppliesOk"
        createInstance="no"
        variable="orderSuppliesResponse">
     <correlations>
        <correlation set="orderCorrelationSet"
               initiate="yes"/>
     </correlations>
   . . .
   </onMessage>
   <onMessage partnerLink="supplier"
        portType="sup:orderSuppliesCallbackPortType"
        operation="orderSuppliesFailed"
        createInstance="no"
        variable="orderSuppliesRFault">
     <correlations>
        <correlation set="orderCorrelationSet" initiate="yes"/>
     </correlations>
   . . .
   </onMessage>
   <onAlarm for="P1M">
        <empty/>
   </onAlarm>
</pick>
```

Providing Web Services Operations (other basic activities)

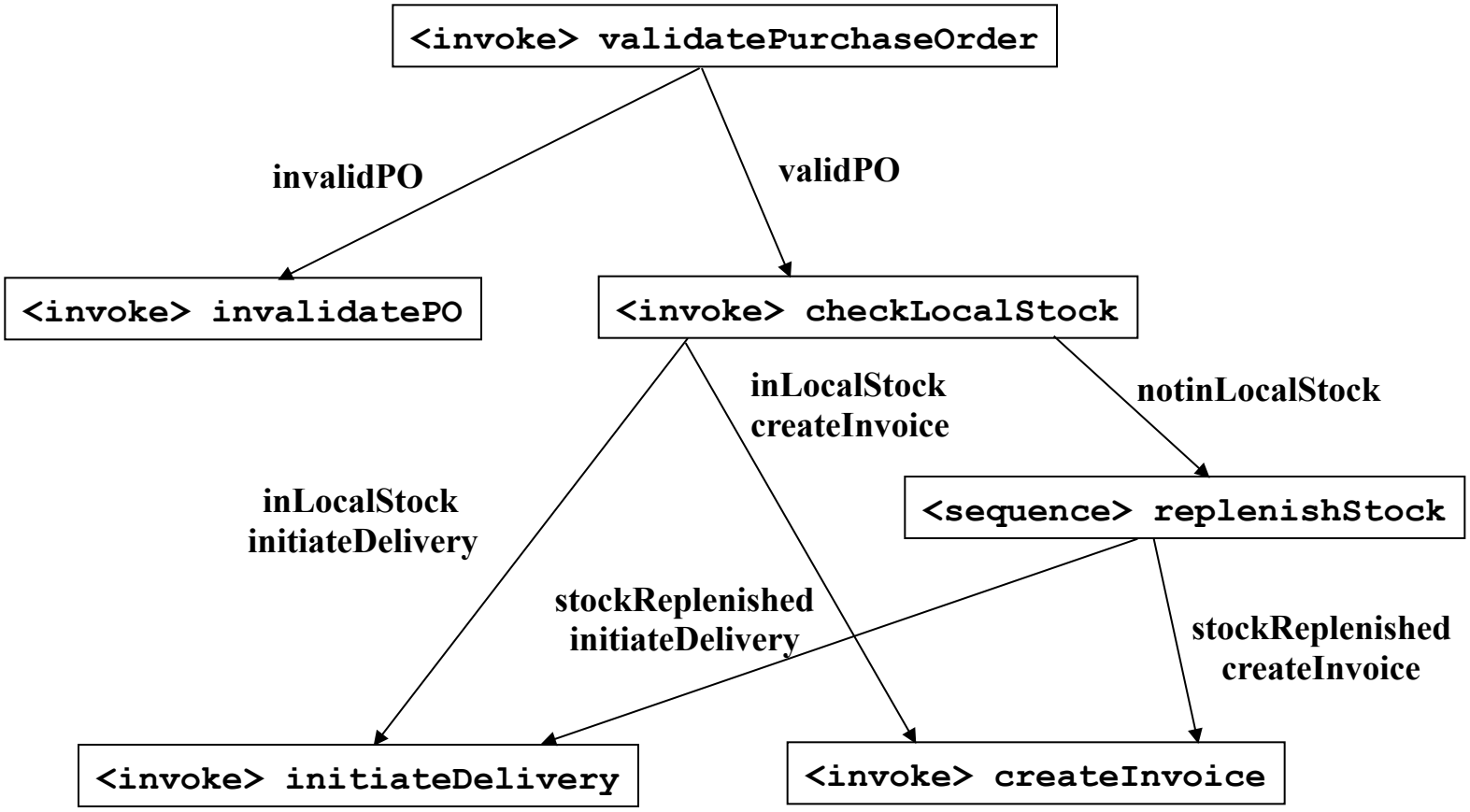
- wait

```
<wait until="2005-02-28T18:00+1:00"/>
<wait for="P3DT10H">
```

- empty

```
<empty/>
```

Providing Web Services Operations (`flow` activity)



Providing Web Services Operations (`links` semantics)

- A link between two activities prescribe an execution order
- It can have associated transition condition
- Fork activity
- Join activity
- Activities of **`flow`** without incoming links starts as soon as the flow is activated

Providing Web Services Operations (**links**)

```
<flow>
  <links>
    <link name="validPurchaseOrder"/>
    <link name="invalidPurchaseOrder"/>
    <link name="inLocalStock_initiateDelivery"/>
    <link name="inLocalStock_createInvoice"/>
    <link name="notInLocalStock"/>
    <link name="stockReplenished_initiateDelivery"/>
    <link name="stockReplenished_createInvoice"/>
  </links>
  <invoke partnerLink="local" portType="skt:skatestownPortType"
          operation="validatePurchaseOrder“ inputVariable="poSubmissionRequest"
          outputVariable="validatePurchaseOrderResponse">
    <source linkName="validPurchaseOrder“ transitionCondition="bpws:getVariableData
          ('validatePurchaseOrderResponse','valid')=true"/>
    <source linkName="invalidPurchaseOrder“ transitionCondition="bpws:getVariableData
          ('validatePurchaseOrderResponse','valid')=false"/>
  </invoke>
  <throw faultName="pos:invalidPO">
    <target linkName="invalidPurchaseOrder"/>
  </throw>
  <invoke partnerLink="local" portType="skt:skatestownPortType"
    operation="checkLocalStock“
          inputVariable="poSubmissionRequest" outputVariable="checkLocalStockResponse">
    <target linkName="validPurchaseOrder"/>
    <source linkName="inLocalStock_initiateDelivery“ transitionCondition=
          "bpws:getVariableData('checkLocalStockResponse','available')=true"/>
    <source linkName="inLocalStock_createInvoice" transitionCondition=
          "bpws:getVariableData ('checkLocalStockResponse','available')=true"/>
    <source linkName="notInLocalStock" transitionCondition=
          "bpws:getVariableData('checkLocalStockResponse','available')=false"/>
  </invoke>
          . . .
</flow>
```

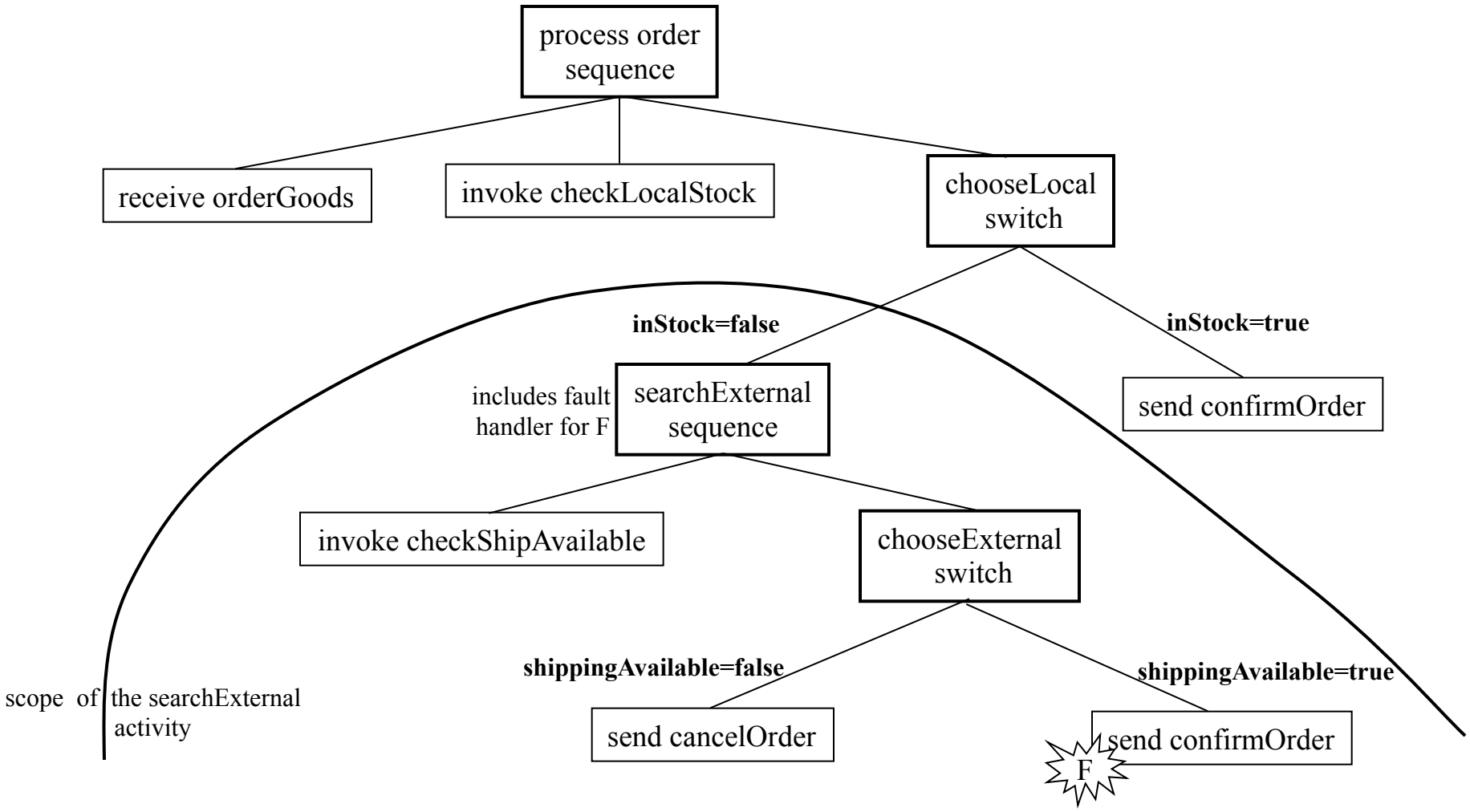
Providing Web Services Operations (**sequence**)

```
<sequence>
  <receive partnerLink="buyer" portType="pos:poSubmissionPortType"
        operation="poSubmission" createInstance="yes"
        variable="poSubmissionRequest">
    <correlations>
        <correlation set="orderCorrelationSet" initiate="yes"/>
    </correlations>
  </receive>
  <flow>
    . . .
  </flow>
  <reply partnerLink="buyer" portType="pos:poSubmissionPortType"
        operation="poSubmission" variable="poSubmissionResponse">
    <correlations>
        <correlation set="orderCorrelationSet" initiate="no"/>
    </correlations>
  </reply>
</sequence>
```

Providing Web Services Operations (**while,switch**)

```
<while condition="bpws:getVariableData('checkLocalStockResponse',
                  'available')=false">
   <sequence>
   . . .
   </sequence>
</while>


<switch>
   <case condition="bpws:getValueData('poSubmissionResponse',
                  'invoice','/totalCost')<100">
   . . .
   </case>
   <case condition="bpws:getValueData('poSubmissionResponse',
                  'invoice','/totalCost')<800">
      <empty/>
   </case>
   <otherwise>
   . . .
</switch>
```

BPEL: Exceptions and transactions



BPEL: Exceptions (fault handling)

- Fault as a result of a Web service invocation

```
<operation name="doSubmission">
   <input message="pos:poSubmissionRequest"/>
   <output message="pos:poSubmissionResponse"/>
   <fault name="invalidPO“
            message="pos:poSubmissionFaultInvalidPO"/>
   <fault name="outOfStock"
            message="pos:poSubmissionFaultOutOfStock"/>
 </operation>
```

- Fault thrown explicitly

```
<throw faultName=“pos:invalidPO">
```

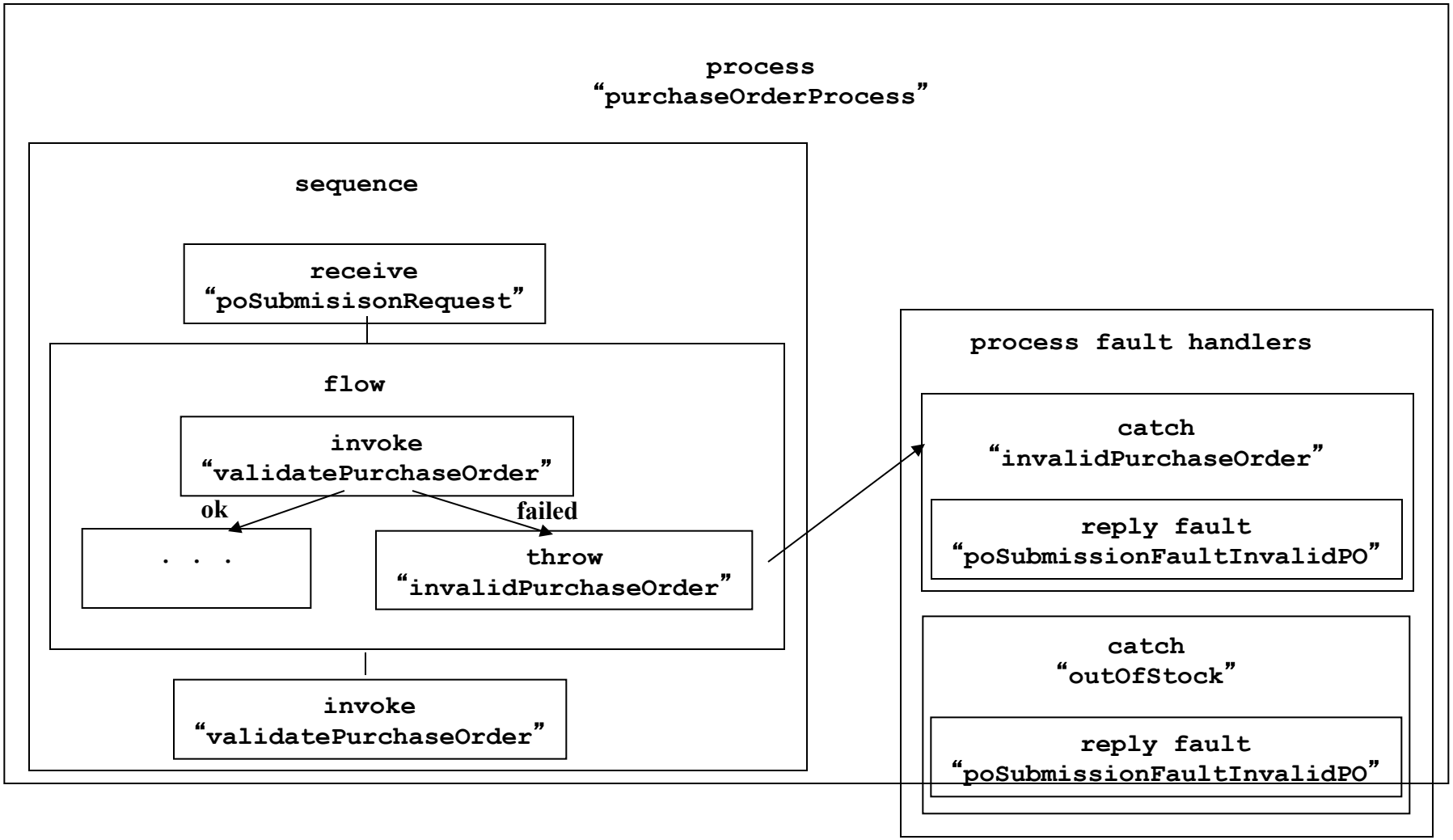
BPEL: Exceptions (fault handling)

- Faults by execution engine (run-time)
  - Incompatible types in an assign activity
  - Fault in enclosing scope
  - Message content doesn't match correlation data
  - Attempt to access an uninitialized part of a variable
  - Attempt to execute reply before corresponding receive was not finished
  - …

BPEL: Exceptions (fault handlers)

```
<faultHandlers>
 <catch faultName="pos:invalidPO">
  <sequence>
   <assign>
     <copy>
     <from variable="poSubmissionRequest" part="purchaseOrder"
               query="/billTo/id"/>
     <to variable="poSubmissionFaultInvalidPO“ part="customerID"/>
    </copy>
    <copy>
      <from variable="poSubmissionRequest" part="purchaseOrder“
               query="/id"/>
      <to variable="poSubmissionFaultInvalidPO“ part="orderNumber"/>
    </copy>
   </assign>
   <reply partnerLink="buyer" portType="pos:poSubmissionPortType"
               operation="poSubmission“ faultName="pos:invalidPO"
               variable="poSubmissionFaultInvalidPO">
     <correlations>
       <correlation set="orderCorrelationSet" initiate="no"/>
     </correlations>
   </reply>
  </sequence>
   . . .
 </catch>
</faultHandlers>
```

BPEL: Exceptions (fault handling)



BPEL: Transactions (compensation handling)

```
<invoke name="checkAndAllocateLocalStock"
        partnerLink="local"
        portType="skt:skatestownPortType"
        operation="checkLocalStock"
        inputVariable="poSumbissionRequest"
        outputVariable="checkLocalStockResponse">
      . . .
   <compensationHandler>
       <invoke name="deallocateLocalStock"
               partnerLink="local"
               portType="skt:skatestownPortType"
               operation="deallocateLocalStock"
               inputVariable="poSumbissionRequest">
   </compensationHandler>
</invoke>
```

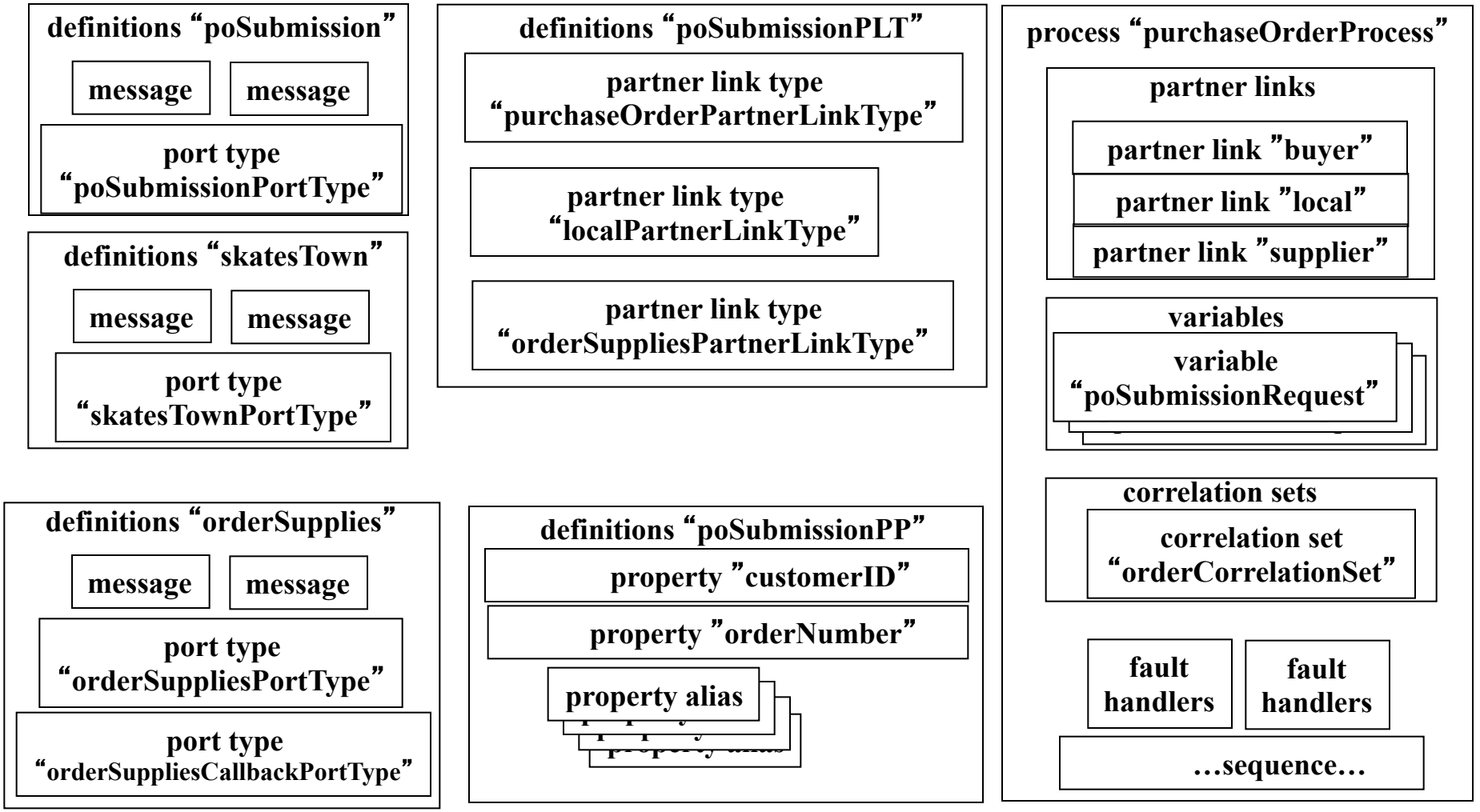
BPEL: Exceptions (event handling)

```
<!-- Event handler definitions -->

<eventHandlers>
 <onMessage partnerLink="buyer"
              portType="pos:poSubmissionPortType"
              operation="cancelPurchaseOrder“
              variable="cancelPurchaseOrderRequest">
   <correlations>
     <correlation set="orderCorrelationSet" initiate="no"/>
   </correlations>
   <terminate/>
 </onMessage>
</eventHandlers>
```

BPEL: Abstract process

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="orderSuppliesAbstractProcess" . . .>
  <partnerLinks>
    <partnerLink name="supplier" partnerLinkType="plt:ordSupPartLT"
                 myRole="seller" partnerRole="supplier"/>
  </partnerLinks>
  <partners>
    <partner name="skatestownSupplier">
       <partnerLink name="supplier"/>
    </partner>
  </partners>
  <correlationSets>
    <correlationSet name="orderCorrelationSet" properties="customerIDorderNumber"/>
  </correlationSets>
  <sequence>
    <invoke partnerLink="supplier"
                portType="sup:ordSuppPortType" operation="orderSupplies">
      <correlations>
        <correlation set="orderCorrelationSet" initiate="no"/>
      </correlations>
    </invoke>
    <pick>
      <onMessage partnerLink="supplier" portType="sup:ordSupCallbackPortType"
                 operation="ordSupliesOk">
        <correlations>
          <correlation set="orderCorrelationSet" initiate="no"/>
        </correlations>
        <empty/>
      </onMessage>
      <onMessage partnerLink="supplier"
              portType="sup:ordSupCallbackPortType"operation="ordSupFailed">
        <correlations>
          <correlation set="orderCorrelationSet" initiate="no"/>
        </correlations>
        <empty/>
      </onMessage>
      <onAlarm for="P1M">
        <empty/>
      </onAlarm>
    </pick>
  </sequence>
</process>
```

SkatesTown Example (all together)



Next lecture

- Web services and Stateful Resources

Text-book, chapter 8